Oikos_Workshop

Chloe Fouilloux

1/19/2021

Hello and Welcome to Oikos Finland 2021! This will be the first part of our coding workshop, my name is Chloe Fouilloux and I am a 2nd year PhD student at the University of Jyväskylä. Besides data visualization, I like being outside, catching frogs, and writing poetry.

Well, enough about me, let's jump into this workshop!

PART 1

Before starting this workshop, you were asked to install some packages. For first time users, there was a brief tutorial on setting your working directory and some quick trouble shooting (see file called Oikos_Package_Install.Rmd). I will quickly reiterate what we went over there:

```
getwd() #Find out your working directory, if necessary

setwd() #Set it to anywhere on your computer. For simplicity, set (/Desktop).

#All that matters here is that your packages are either marked as
#"downloaded" or "unpacked"

install.packages(c("ggplot2", "ggforce", "dplyr", "ggsignif", "png", "Hmisc"))
```

First thig first, let's load some packages. For those who have already done this, you won't need to do this a second time.

```
#If this is your first time loading these packages, you have to install them first.
#YOU ONLY HAVE TO INSTALL PACKAGES ONCE.

#First, let's load some packages!
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
library(ggforce)
```

```
## Warning: package 'ggforce' was built under R version 3.6.2
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggsignif)
library(png)
library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 3.6.2
## Loading required package: lattice
## Loading required package: survival
## Warning: package 'survival' was built under R version 3.6.2
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

Great. so to avoid errors and troubleshooting during our workshop, we have decided to work from data that already exists in base R and doesn't require any downloading.

```
#You can actually see all of them by just entering the following:

data()
#These are all the datasets that R automatically provides for you,
#and is great for practicing!

#Today we will be using two data sets, please load them now:

data("warpbreaks")
data("iris")

#Don't forget the quotation marks!
#You will find the datasets waiting for you under the "Values" column --
#should have something along the lines of <Promise>, click on them,
#and they will be loaded into your working data.
```

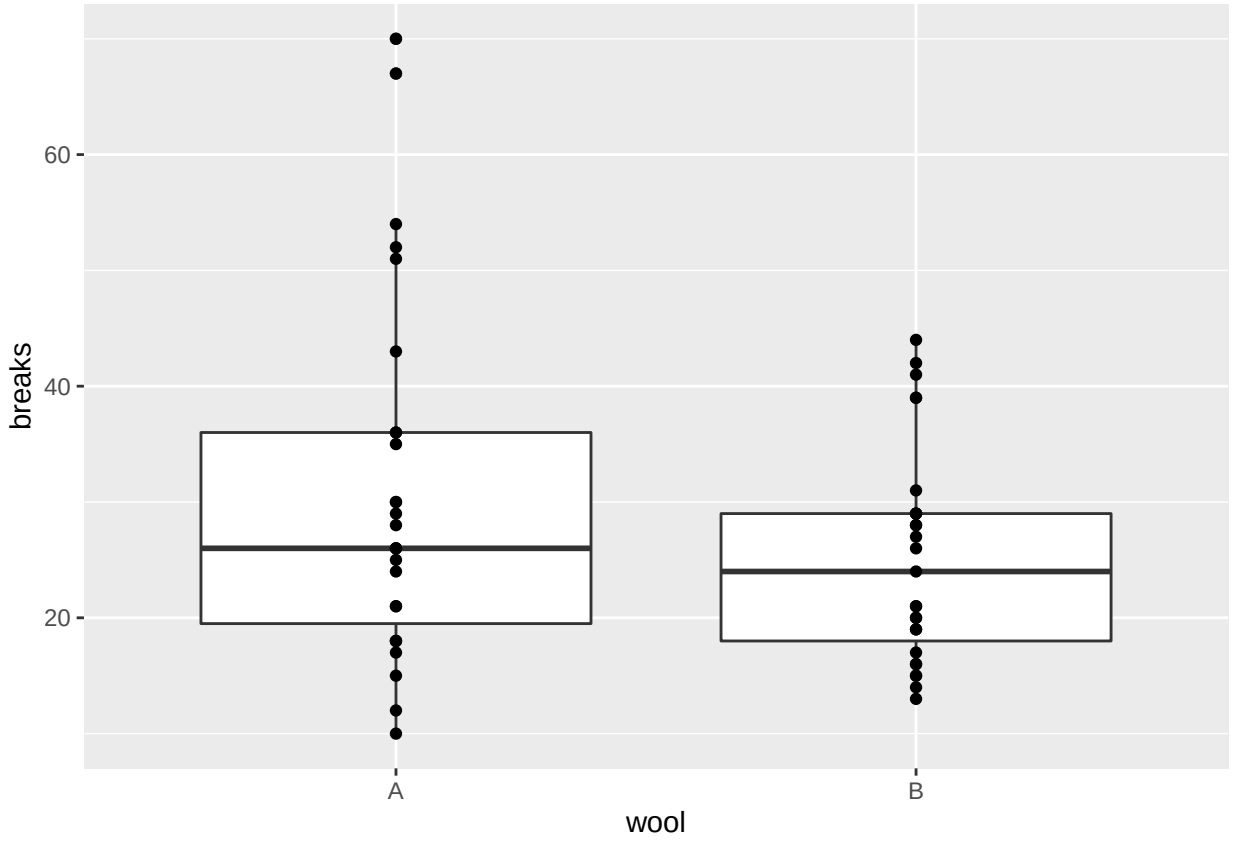
Okay, we've got our packages, we've got our data.

First let's look at our "warpbreaks" data. We've got two factors and one numerical variable. This is important to know when we start thinking about how we want to visuliaze data.

Looks like we're ready to plot!

```
#First, I always like to just look at the data and see what could
#potentially be interesting to measure.

#This will of course will be more intuitive for you when you know
#your own data and what questions you have.
```

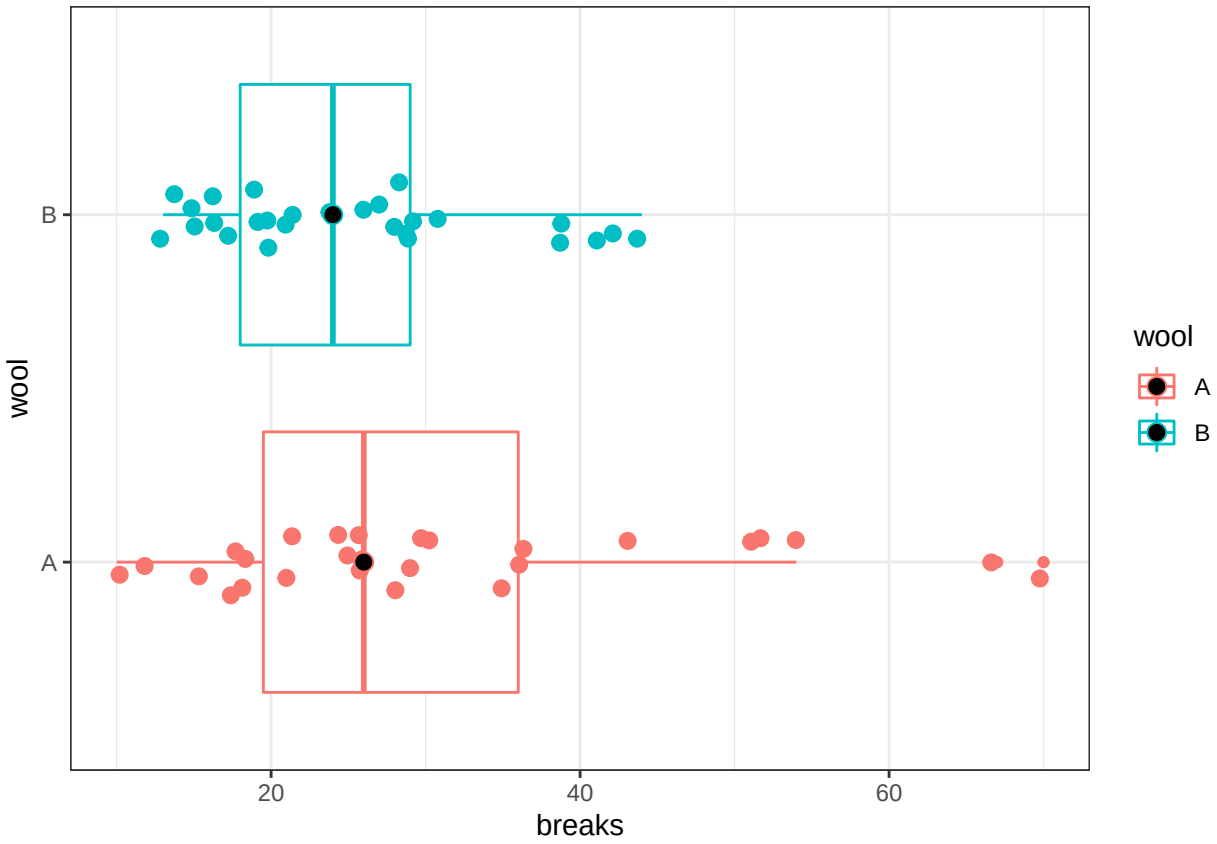
```r
#For now, let's look at this wool data. . . who likes to knit ? :)

#Alright, let's jump into ggplot.
ggplot(warpbreaks, aes( x = wool, y = breaks))+
  geom_boxplot()+
  geom_point()
```



So, this is a very basic boxplot. What could be improved upon here? Well, the points are hard to distinguish, the plot is kind of bland, and it's pretty hard to say for sure that anythere of interest is going on. . .

Let's first talk about points/jitter and introduce the concept of stat_summary

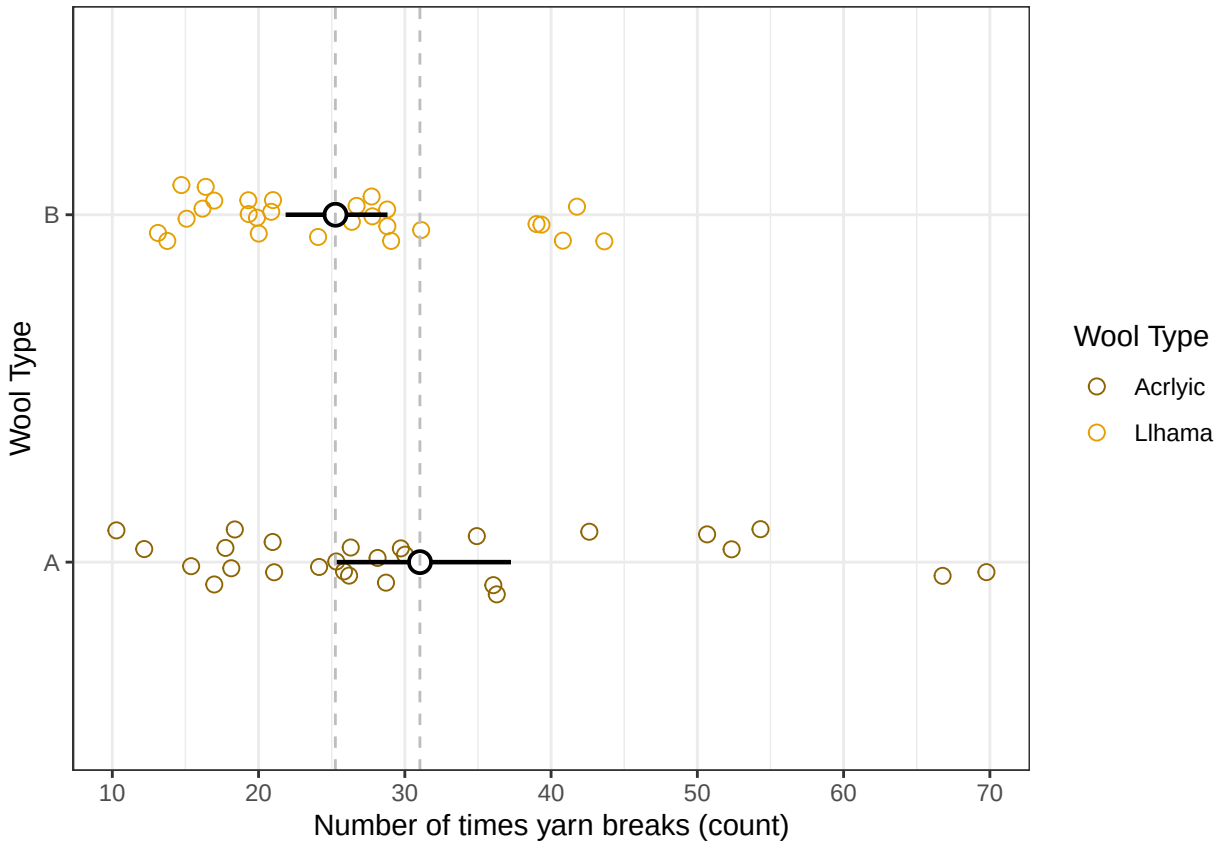
```r
ggplot(warpbreaks, aes( x = wool, y = breaks, color = wool))+
  geom_boxplot()+
  geom_jitter(width = 0.1, size = 2.5) +
#this introduced controlled noise into your data to better visualize points
  stat_summary(fun = "median", geom = "point", shape = 21, size = 3, fill = "black") +
#how aobut emphasizing some key stats?
#fill vs color depending on shape, see how I made shape black?
#I could also color it by our data if that was interesting.
  #fun = min, max, mean, median
  #geom = pointrange, crossbar, errorbar
  coord_flip() + #switch x and y
  theme_bw() #get that grey background out of there
```



That's all well and good, but admittedly, the plot looks a little boxy. . . and I am still not sold on the colors. Also, boy oh boy look at that legend! What the heck is going on.

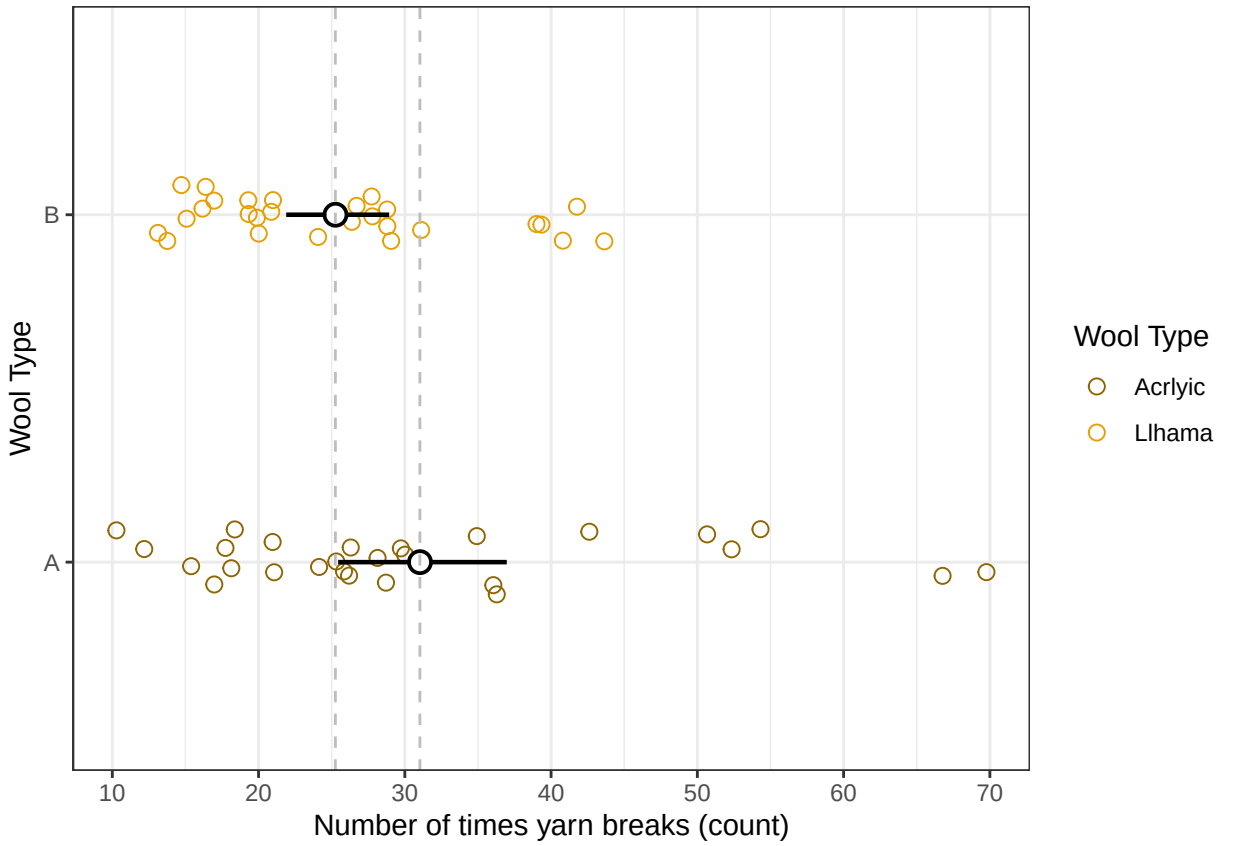
```
#Version Number 3

a<-ggplot(warpbreaks, aes( x = wool, y = breaks))+
#Let's get those boxplots out of here for now
  geom_jitter(width = 0.1, size = 2.5, shape = 1, aes(color = wool)) +
#let's talk about shapes!
  stat_summary(fun.data = mean_cl_boot, #let's look at mean instead of median
               geom = "pointrange", # could also do crossbar, errorbar
               shape = 21, #consult shape guide on powerpoint
               fill = "white",
               size = 0.8)+
#Great we can really make the stat summary quite complex if we wish,
#now, let's think about our axis labels
  scale_y_continuous(breaks = seq(0, 70, by = 10))+
#Let's get some more numbers on that xaxis to make mean comparisons easier
  xlab("Wool Type")+
  ylab("Number of times yarn breaks (count)") +
#again, don't forget the quotes for axis and variable names!
#fill vs color depending on shape
#----
  #let's work with color, anything under the sun!
  #https://encycolorpedia.pl/fddc8b
 #Let's also perfect our legend! I have completely made up these names, but I think
 #they're cute.
  scale_color_manual(values = c("#8d6502", "#e69f00"),  #note color vs fill
```

```
                    name = c("Wool Type"), #Legend Title
                    labels = c("Acrlyic", "Llhama"))+ #Factor title2
#now, quickly, I would like to introduce you to some annotation tricks
#I don't think they are "perfect" here, but I think they could be useful in your
#own coding life and it's a good thing to have in your coding arsenal.

#hlines go x axis, v lines go y axis. If I want my lines to go through the
#boot strapped mean, I do the following. (we will talk about how to access this
#value later)
  geom_hline (yintercept = 25.25, linetype = "dashed", color = "grey")+
  geom_hline (yintercept = 31.03, linetype = "dashed", color = "grey")+

#I also think an underused trick is flipping axes to compare values. Let's do that again
#here.
  coord_flip() +
  theme_bw()



#Great job! We completed the first plot.
#Now, let's look into this computation by stat_summary.
#The first think we have to do is save our plot. Let's call it "a".

#we do this by adding "a <- " in front of our plot. This ggplot is now saved as an object.
#(you should see it in your global environment)
#how to find data from mean_cl_boot
#first, save ggplot as an object
a
```



So, let's explore the maths behind our first plot a little more.

```
a
```




```
#We use a base function called ggbuild. Again, we must save this information,
#let's call it b!
b<- ggplot_build(a)

#here we can see the information stored under "y" with the confidence
#lmits as ymin and ymax.

#Let's go through navigating this path together!
b[["data"]][[4]]
```

```
##   yintercept PANEL group colour size linetype alpha
## 1      31.03     1    -1   grey  0.5   dashed    NA
```

PART 2

Alright, so you guys have made it through the first part of the data visualization workshop. Are you in **love** with this kind of stuff yet? Isn't it so much fun?

If you aren't passionate about making your statistical nightmares into dreamy plotscapes, then maybe you just need a little bit of a challenge.

Here we go for part two.

```
#Let's call our iris data set (you've already loaded it,
#but here's a good excuse to remind yourself)
data("iris")

#Like with the wool data set, let's look at it a little bit to see what's going on.
```

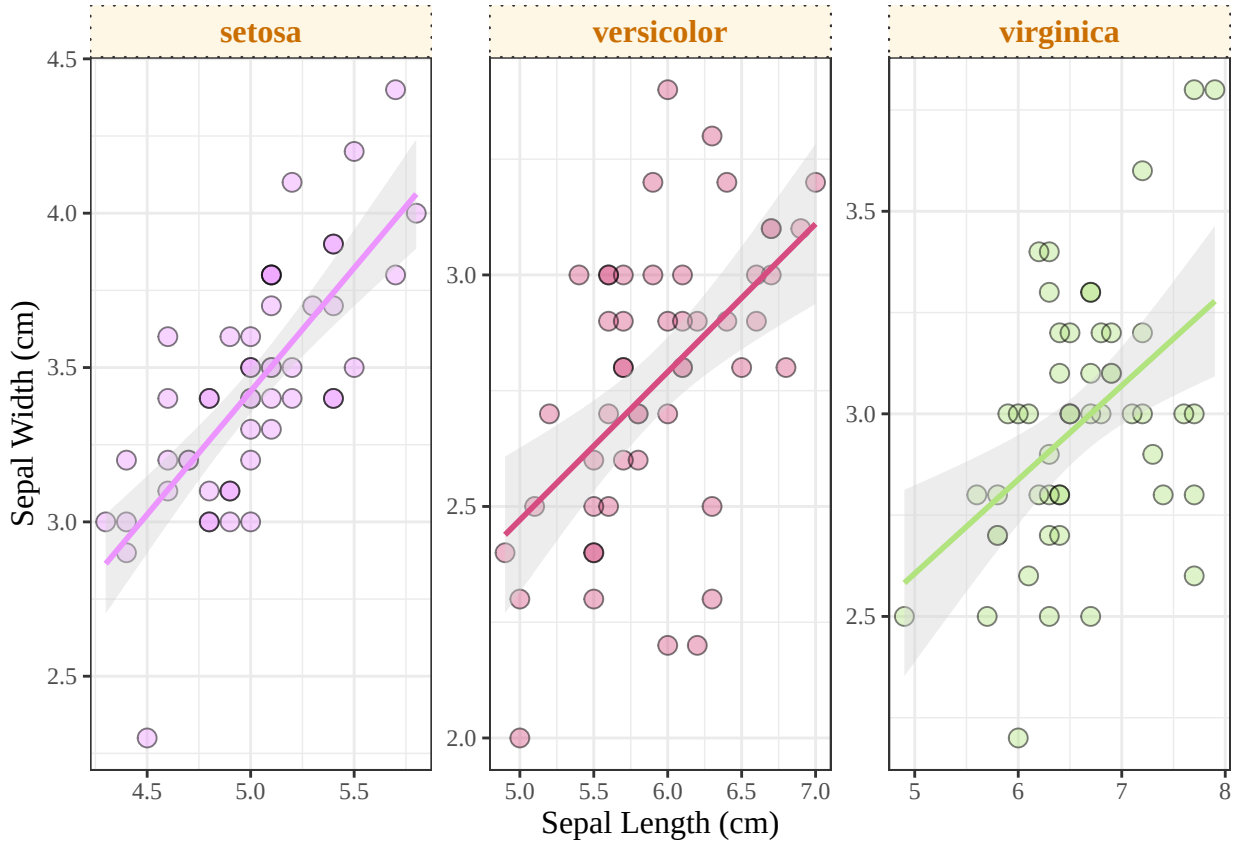
```
ggplot(iris, aes( x = Sepal.Length, y = Sepal.Width, color = Species))+
  geom_point() # I like this one bests´data
```


```
#Hmmm, definitely looks like these species have different lengths of sepals.
```

Coolio, we've foind something of potential interest here, let's looking.

```
ggplot(iris, aes( x = Sepal.Length, y = Sepal.Width, color = Species))+
  geom_point(size = 3, shape = 16, alpha = 0.4) +
  geom_point(size = 3, shape = 1, alpha = 0.5, color = "black")+
#did you know you can keep adding layers of points, of various sizes
  ylab("Sepal Width (cm)")+ #let's label some axes
  xlab("Sepal Length (cm)")+
#look at what happens when I switch between color and fill

#Alright, now let's take this graph from a scatter plot to something challenging and functional!
#Let me introduce you to stat_smooth, which is a function that makes regression lines in your plot!
  stat_smooth(method = "glm", fill= "lightgrey", #lm, loess
              formula = y~ x, fullrange = F) +  #log(x), poly(x, 3); fullrange F/T
#With great power comes great responsibility!
#I now want to make a very important rule very clear! This has to do with scales
#and "zooming" into parts of your plot, which you will inevitably do in your career with ggplot.
#Let's imagine we want to zoom in to emphasize the setosa species.
              #coord_cartesian(xlim = c(5,5.5))
              #xlim(5, 5.5)
#Of course, with the second option, you havea ton of error messages, but the code
#still runs, so you must always be careful and read your error messages!
```

```
  facet_wrap(~ Species, scales = "free")+
#Alright, so often in ggplot, no one ever messes with the base theme,
#but it's completely customizable!
#The font, the background, the colors, everything!
#Now, let's try really messing everything up here. :=)
  theme_bw(base_family = "Times") + #change the text everywhere
  theme(
    legend.position = "none",
    axis.title =
      element_text(size = 12),
    strip.background = element_rect(fill = "#fff7e5", linetype = "dotted"),
    strip.text= element_text(
                              size = 12,
                              face = "bold",
                              color = "#ca6e00")) +
  scale_color_manual(values = c("#ec94ff", "#d64b80", "#b1e37e"))
```




```
################
#### BONUS ---- ##
################

#Using dplyr to make gramatically correct plots!

#You might be also wondering about how to change
#the species names to have captial letters!

#Make sure you have dplyr loaded for this
```

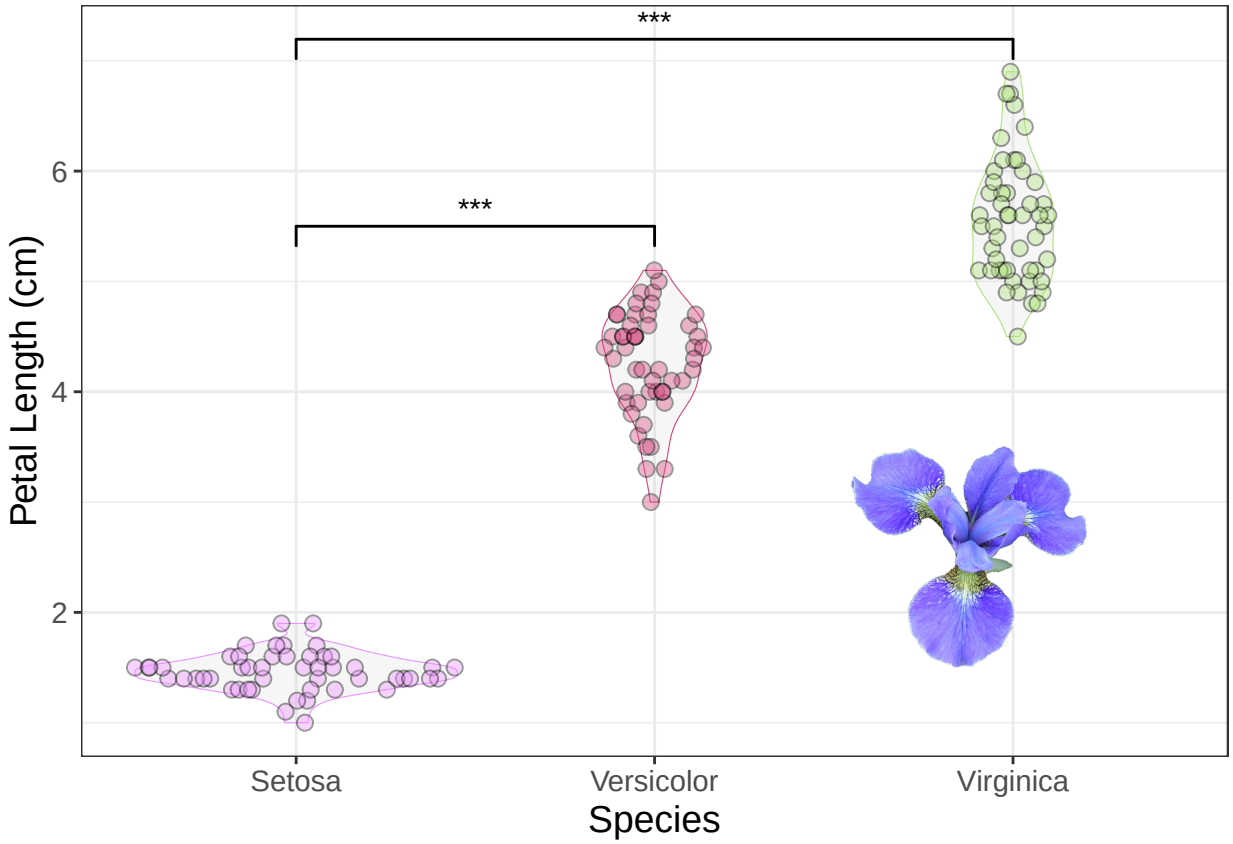
```
iris$Species<- recode(iris$Species,
                      setosa = "Setosa",
                      versicolor = "Versicolor",
                      virginica = "Virginica")


#After you run this, you can run the plot again.
#Usually, we would put all of the dplyr/tidyr
#modifications at the beginning of the script
#(that way you only run everything once),
#but it's here for teaching purposes.
```

PART 3

Well, I hope you've learned something through all of this! The next plot it probably the highest probability of people getting lost, but hopefully the first two plots have taught you enough to either be able to come and revisit this later and figure the code out or to follow along the first try!

```
##################################
#Adding pictures to your graph! ----
#Read the last chunk if you need help setting up your working directory
#and loading pictures
##################################
img <- readPNG("iris1.png")

#save your image, the title is whatever you named it. Mine is called iris1
#THIS SHOULE BE SAVED IN YOUR WORKING DIRECTORY! :)
#discuss "::" syntax
i1 <- grid::rasterGrob(img, interpolate = T)

#See, just two lines of code! Now, add this to your original ggplot in annotation_custom

#We've been looking at sepals, so what not look at some petals!
ggplot(iris, aes( x = Species, y = Petal.Length))+
  geom_violin(fill = "lightgrey", aes(color = Species),
  #NOTE: see I moved color here, try moving it around and see what happens!
              size = 0.2, alpha = 0.2) +
  #Let's play with some violin plots here, note that size in this case are line thickness
  #When we just look at the distributions, what can we already see?
  geom_sina(alpha = 0.4, size = 2.5,
            shape = 21,
            aes(fill = Species)) +
#Sina plots are a lot like jitter plots except that they are controlled by density!
#Which makes them a perfect pair to go with density plots.

#Alright, let's add some singificant comparisons!
#**NEVER DO THIS WITHOUT HAVING STATISTICAL CONFIRMATION;
# BELOW IS FOR TEACHING PURPOSES ONLY**
#Run list with capital letters if you did the bonus in part 2.
#if you skipped it, make sure
#you keep them the way they are originally coded in the dataset.
  ggsignif::geom_signif(comparisons = list(c("Virginica", "Setosa")),
                        map_signif_level = TRUE)+
```

```
ggsignif::geom_signif(comparisons = list(c("Versicolor", "Setosa")), y_position = 5.5,
                      map_signif_level = TRUE)+
#sina functions where the width
#of the jitter is controlled by the density distribution of the data
theme_bw()+
theme(legend.position = "none",
      text = element_text(size = 14))+
ylab("Petal Length (cm)")+
scale_fill_manual(values = c("#ec94ff", "#d64b80", "#b1e37e"))+
scale_color_manual(values = c("#ec94ff", "#d64b80", "#b1e37e")) +

#Alright let's take a coffee break before we go attacking adding images into plots.
#Find a picture of an iris flower, and save it in your working directory. (see below)
annotation_custom(i1, #this is my picture!
                  ymin = 1.5,
                  ymax = 3.5,
                  xmin =1.75,
                  xmax = 4)
```



Alright, so for adding pictures, you need to put the picture file (.png) in your working directory. First, let's find out where that is on your computer! (Usually it's your desktop if you haven't done anything to it)

Here we see my working directory if a folder on my Desktop called Oikos (yay! Oikos 2021 wooooooo!). So, I would need to drag my image there.

Now, check your working directory.

Where is it? If you have something really weird, just set it to your desktop for now. Once that's taken care

of, go to the previous chunk and re-run the code.

Questions? Color choice resources: > https://encycolorpedia.pl/fddc8b > https://paletton.com/#uid=34H0u0kllll8EyBe+r6rGf-B39O

Read more about stat summary calculations and options here: > https://rstudio-pubs-static.s3.amazonaws.com/3358_2def158be48047a6944bfcc60ea08f1b.html

Read more about the significance package (ggsignif) here : > https://cran.r-project.org/web/packages/ggsignif/vignettes/intro.html